Session 501

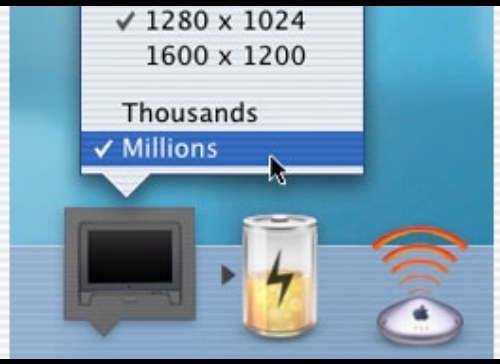
Java Virtual Machine

Blaine Garst
Manager, Java Runtimes Technology

What You'll Here About

- JVM Basics
- What HotSpot is all about
- Apple's JVM Extras
- Java Developer Preview 1
- Q/A

JVM Basics



What Does a JVM Do?

- JVM = Java Virtual Machine
- Manages threads of bytecode execution
- Collects unused Java objects
- Shifts control to-from “native” code

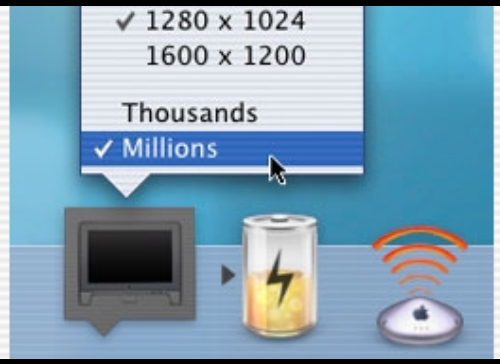
Where Is Apple's Java?

- JDK 1.3
  - **/System/Library/Frameworks/JavaVM.framework**
  - **/usr/bin/java***
  - **/Library/Java/Home**
- Cocoa Java
  - **/System/Library/Java/***
  - **/usr/lib/java/***

Your Java Goes Here:

- /Library/Java/Home
  - /bin
  - /lib
    - /ext
    - /fonts
    - /images
    - /*.properties
    - /security
    - /*.idl
- /Applications

HotSpot



Apple's JVM

- Start with Sun's HotSpot technology
- Adapt it to Darwin (Mach/FreeBSD)
- Write the PPC interpreter
- Write the "client" PPC compiler
- Make it better

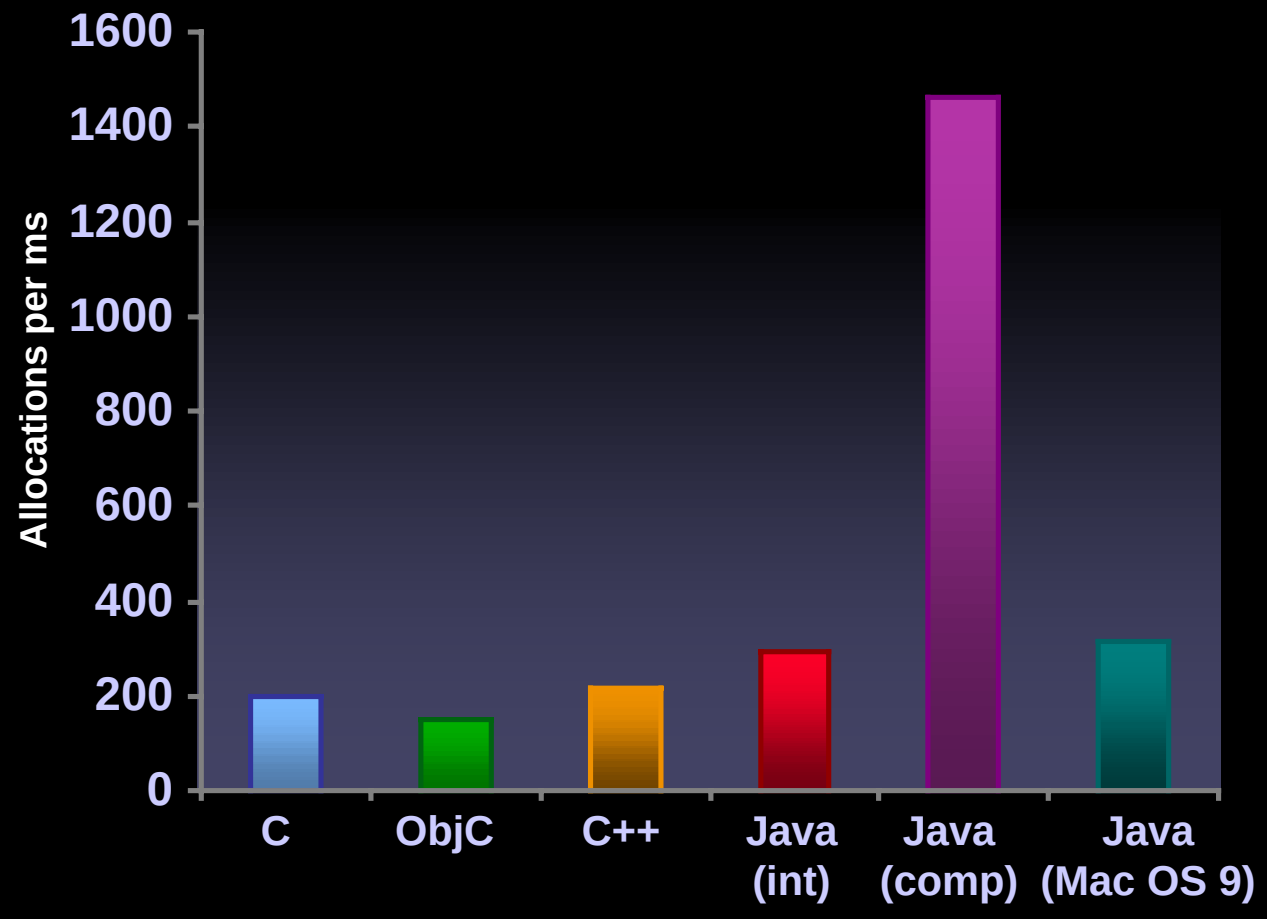
Sun's HotSpot

- 700 files, 200,000 lines of C++ code!
- Fast interpreter—generated on launch
- Speedy bytecode compilation into fairly good code
- Efficient **synchronize** implementation
- Compacting Generational Garbage Collection

Allocation MicroBenchmark

- Several threads in allocate n, free n race
- Compare C, C++, ObjC (Cocoa), Java implementations
- Run it on dual MP
- Note: about Microbenchmarks
  - Atypical usage pattern
  - Your mileage will be less

Allocation MB Results



Garbage Collection @ 41

John McCarthy	**1960**	**Mark & Sweep**
Marvin Minsky	**1963**	**Copying**
Dave Ungar	**1984**	**Generational** **+ 1000 Other Papers**

- Garbage Collection”,1996, Wiley
  - Richard Jones
  - Rafael Lins

Generational Collecting

- "Most objects die young"
- Split memory into generations
- Allows minimization of:
  - CPU cycles to allocate new
  - CPU cycles to remember old
- Use a write-barrier for x-gen stores

Sun's HotSpot GC

Eden	ptr = mem mem += size	Baby
New	2-Space Copy	Child
Tenured	Mark & Sweep or Train	Adult
Permanent	Mark & Sweep	Leaders

Sun's HotSpot GC

Eden	ptr = mem mem += size	Baby
New	2-Space Copy	Child
Tenured	Mark & Sweep or Train	Adult
Permanent	Mark & Sweep	Leaders

Sun's HotSpot GC

Eden	ptr = mem mem += size	Baby
New	2-Space Copy	Child
Tenured	Mark & Sweep or Train	Adult
Permanent	Mark & Sweep	Leaders

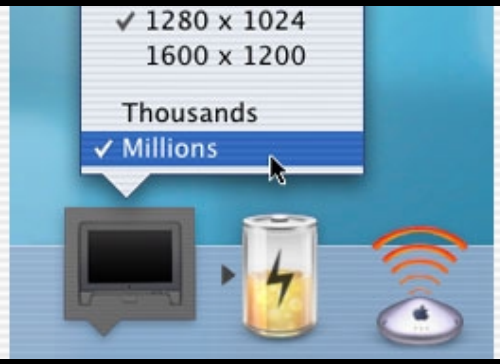
Sun's HotSpot GC

Eden	ptr = mem mem += size	Baby
New	2-Space Copy	Child
Tenured	Mark & Sweep or Train	Adult
Permanent	Mark & Sweep	Leaders

Sun's HotSpot GC

Eden	ptr = mem mem += size	Baby
New	2-Space Copy	Child
Tenured	Mark & Sweep or Train	Adult
Permanent	Mark & Sweep	Leaders

Apple JVM Extras



Make it Better

- Better C/Obj-C language integration
  - Enables more APIs for you
- Better Performance
  - More with less Memory
    - Client scalability is ratio Apps: Memory
  - Faster Launch Times
  - Faster Running Time

Better Language Integration

- JNI is the standard
  - Required for precise GC
- JDirect for Carbon/QT Java
  - Dynamic JNI stub C code generator
- JavaBridge for Cocoa Java
  - bridget stub C and Java generator tool
  - Allows subclassing of ObjC in Java

JDirect 3 Example

```
import com.apple.mrj.jdirect.Linker;
public class Prime {
   // this call to new Linker() creates and installs the native methods
   public static Object linkage = new Linker(Prime.class);

    // this string tells Jdirect where to find the native code
    public static final String JDirect_MacOSX = “prime.dylib”;

    // the C function you want to call
    public static native long ComputePrime(short n);
}
```

Cocoa Example

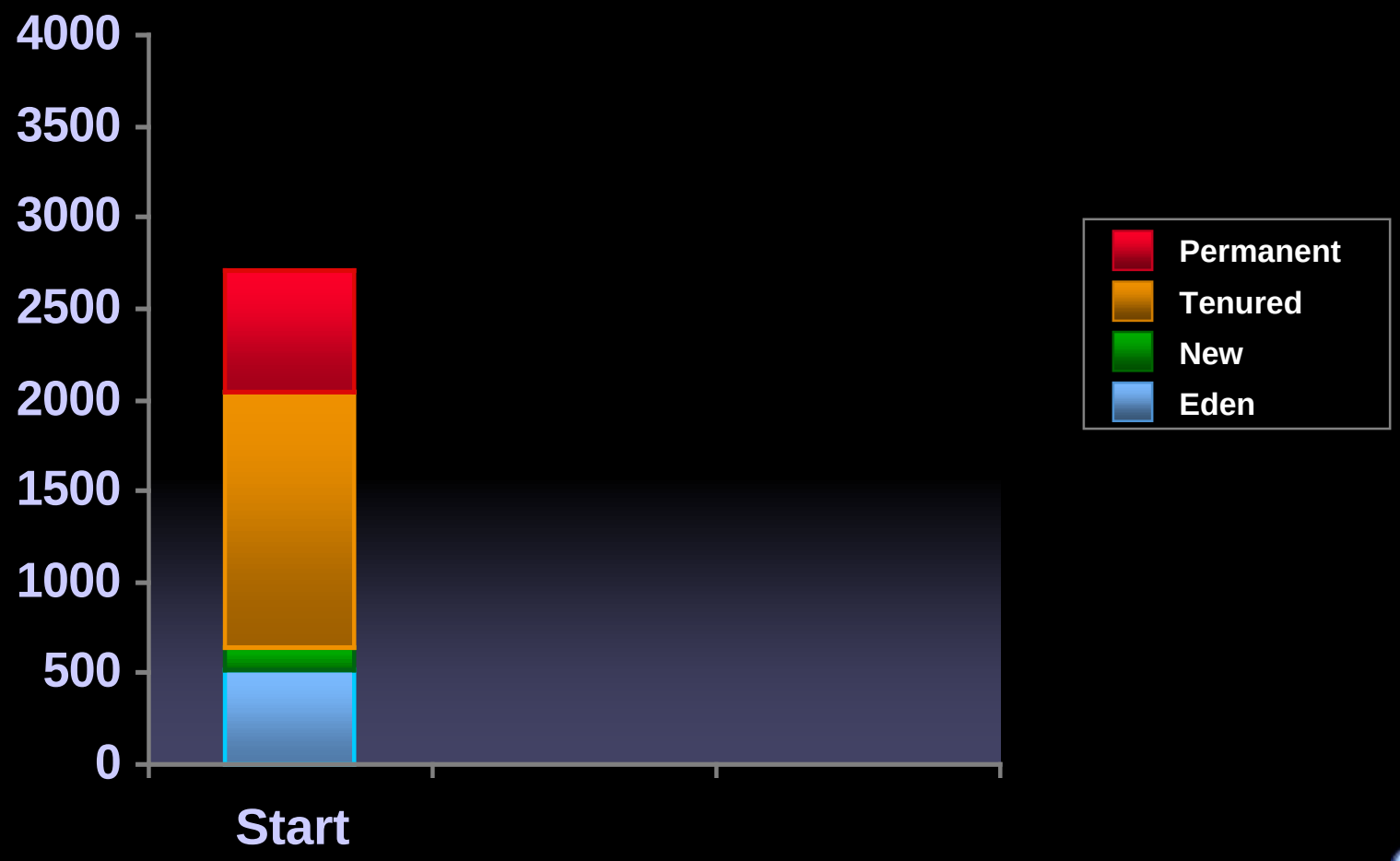
```
import com.apple.cocoa.foundation.*;
public class DateFormatter {
   public static void main(String args[]) {
     String f = “%B %e %H:%M”;
     NSGregorianDateFormatter df = new NSGregorianDateFormatter(f, true);
     try {
         NSDate d = (NSDate)  df.objectValueForString(“next Tuesday at dinner”);
         System.out.println(d);
         System.out.println( df.stringForObjectValue(d));
    }  catch (FormattingException e) { System.err.println(“format error”);  }
  }
}
```

Better Performance

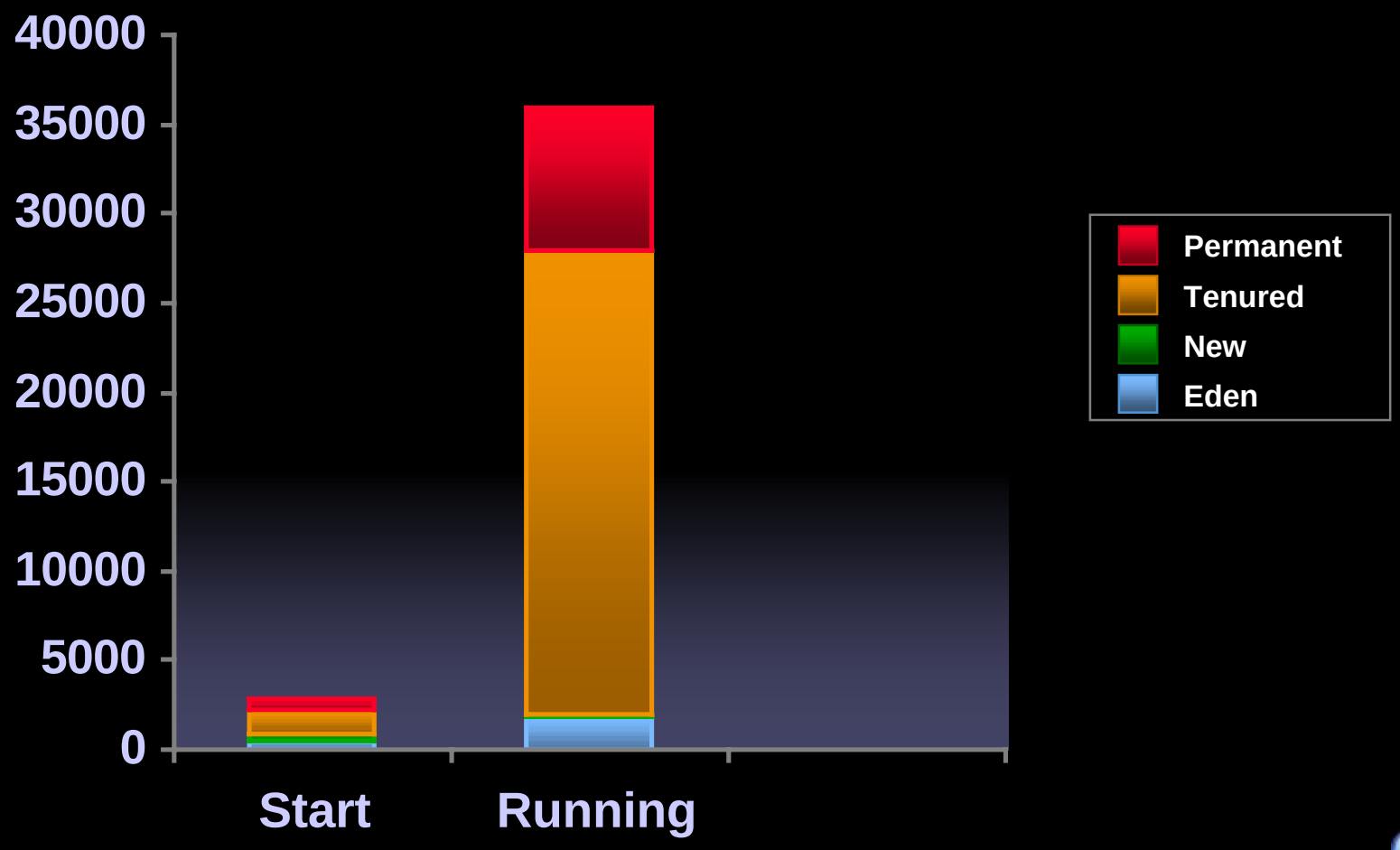
How?

- Memory is evil
- Steal good ideas
- Shared libraries of C code and data
  - Machine code is dominant
  - Constant C strings important
- What about sharing jar files?

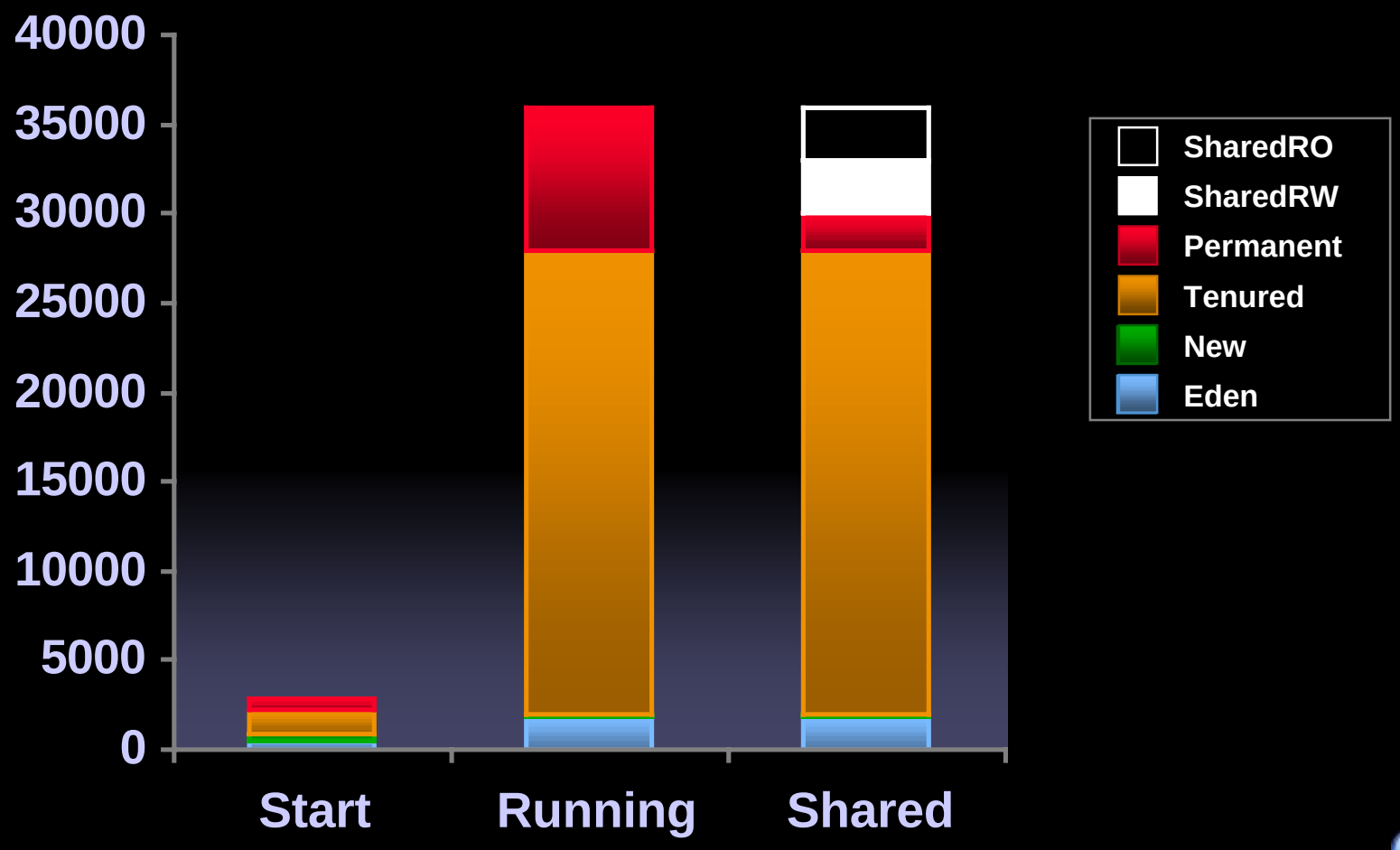
Initial Memory Configuration



Running Memory Configuration



Shared Memory Configuration



Sun's HotSpot GC

Eden	ptr = mem mem += size	Baby
New	2-Space Copy	Child
Tenured	Mark & Sweep or Train	Adult
Permanent	Mark & Sweep	Leaders

Apple's HotSpot GC

Eden	ptr = mem mem += size	Baby
New	2-Space Copy	Child
Tenured	Mark & Sweep or Train	Adult
Permanent	Mark & Sweep	Leaders
Shared	None!	Immortal

The Shared Generation

- "Some objects never change or die"
  - Bytecodes
  - Constant strings
- Process standard jar contents once
  - Load often used standard classes
  - Don't execute anything!
  - Use GC to pack them to Shared Gen
  - Write it to disk
- Map it in on next start up and go!

Shared Generation Benefits

- No GC cycles for Shared Gen*
- Rarely read standard jars!
  - Saves cycles and memory and disk I/O
  - Hot start always faster
- Packed ordered meta-data saves pages!
- Sharing saves 3 to 6 meg for Apps 2->n!

Shared Generation Caveats

- Can't share application jars
- First start is slower
- Interpreter slightly slower
- Don't change BOOTCLASSPATH
  - JBuilder tip:
    - **addskippath ../lawt.jar**
    - **JBuilder5/OpenTools/JSA.config**

Shared Generation Directions

- Improve hot start launch time
- Eliminate first start penalty
- Extend to **'lib/ext'** jars
- Finer grained meta-data ordering
- Finish GC work on **rw** shared section
- Share more runtime data structures

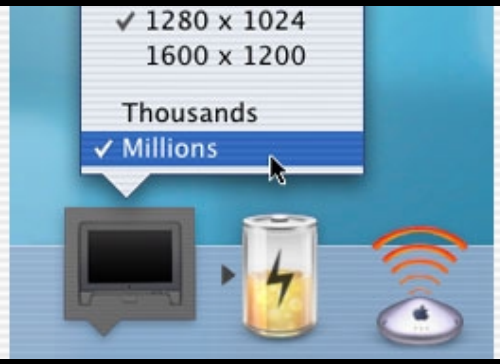
Non-Directions

- Don’t share compiled bytecodes
  - Its only 1-2 Meg really tight code
  - Hard to share and stay good and fast
- Don’t ever use shared RW memory

Shared Generation Status

- In Mac OS X 10.0.0!
- JDK 1.4 (Merlin) feature request
  - *Code provided to Sun*
- Interaction with Sun very positive
  - Feedback goes both ways

Java Developer Preview 1



JVM in Java DP1

- Two fixes since Mac OS X 10.0.0
  - MTF now days instead of hours
- For debugging and profiling:
  - HotSpot 2.0 from JDK 1.3.1!!!

HotSpot 2.0 Benefits

- Client From JDK 1.3.1
- Debugger (JVMDI) speed fix
- Profiling (JVMPI) works
- Better register allocator when compiling bytecodes

Apple Java DP1 Extras

- G4 accelerations
- Optimized instanceof
- Better register usage
  - Floating point
- (Sharing)
- (Even better startup time)
- (Per-thread allocation pools)

Playing With HotSpot 2.0

- Use it casually:
  - **java -hs1_3_1 ...**
- Use it all the time:
  - Slam **libjvm.dylib symlink**
- Tell us about it!

Java Developer Preview Availability

- Sign up at **developer.apple.com**
- Goto **connect.apple.com** and download
- Preserves:
  - **JavaVM.framework/1_3**
  - **/Library/Java/Home/lib**
  - **/Library/Java/Home/bin**
- Discuss:
  - **http://developer.apple.com/java**
  - Check out **java-dev**

Roadmap

502 Wrapping Mac OS APIs in Beans	Room J2 **Tue., 5:00pm**
503 Java Development Tools	Room A1 **Wed., 10:30am**
504 Java Performance	Room C **Wed., 3:30pm**
505 JBuilder	Civic **Wed., 5:00pm**

Roadmap

506 Mac OS X Features for Java Apps	Room A2 **Thurs., 9:00am**
507 QuickTime for Java	Civic **Thurs., 2:00pm**
508 Java Graphics	Civic **Thurs., 3:30pm**
509 Java Security	Room J2 **Fri., 9:00am**
Java Feedback Forum FF117	J1 **Fri., 10:30am**

Who to Contact

Alan Samuel
Java Technology Manager
Apple WorldWide Developer Relations
bluker1@apple.com

Q&A

Mac OS X Java Runtime Technology Team

http://developer.apple.com/events/urls.html

WWDC

Worldwide Developers Conference 2001